On the Satisfiability of Constraints in Workflow Systems

Jason Crampton

Technical Report
RHUL–MA–2004–1
24 May 2004

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Abstract

The specification and enforcement of authorization policies such as separation of duty and binding of duty in workflow systems is an important area of current research in computer security. We introduce a formal model for constrained workflow systems that incorporate constraints for implementing such policies. We define an entailment constraint, which is defined on a pair of tasks in a workflow, and show that such constraints can be used to model many familiar authorization policies. We show that a set of entailment constraints can be manipulated algebraically in order to compute all possible dependencies between tasks in the workflow. The resulting set of constraints form the basis for an analysis of the satisfiability of a workflow. We briefly consider how this analysis can be used to implement a reference monitor for workflow systems.

1 Introduction

The term "workflow" was first used to describe the scheduling of jobs on a mainframe computer by the operating system [6]. The term is now used generically to describe an ordered collection of tasks T each of which needs to be performed by an appropriate agent. Typical examples of workflows include purchase order processing [4], the handling and refereeing of papers in electronic journals [2], and the processing of tax refunds [3]: indeed, practically any complex business process can be modelled as a workflow.

However, there remain significant challenges to be resolved before we see the widespread use of sophisticated commercial computerized workflow management systems. Of particular interest to the security community is the problem of authorizing users to execute tasks within a workflow while enforcing constraints such as separation of duty on the execution of those tasks [2, 3, 4, 7, 9, 14].

Role-based access control (RBAC) is a natural paradigm to apply to authorization in workflow systems because of the correspondence between tasks and permissions. In recent years, a considerable amount of work has been done on the use of RBAC to support access control in workflow systems [1, 3, 7, 14].

However, a role-based model alone is not sufficient to meet all the authorization requirements of workflow systems such as *separation of duty* constraints and *binding of duty* constraints. Separation of duty requirements exist to prevent conflicts of interest and to make fraudulent acts more difficult to commit. A simple example of a separation of duty constraint would be to require two different signatures on a cheque. Binding of duty constraints require that if a certain user executed a particular task then that user must also execute a second task in the workflow. Additionally, *cardinality* constraints are used to specify that a particular task must be performed a given number of times, optionally by a given number of different users.

In the context of workflow systems, a separation of duty requirement may be that two tasks are performed by different users (or different roles). There exist several schemes and models in the literature for specifying separation of duty constraints [2, 3, 4, 7, 9, 14]

and cardinality constraints [3] in workflow systems. We also note that that order-based separation of duty in role-based systems [8] can, and probably should, be studied in the context of workflows.

We model a workflow specification as an ordered set of tasks and a constrained workflow specification as an ordered set of tasks together with a set of authorization constraints. A (constrained) workflow authorization schema is a (constrained) workflow specification augmented by authorization information. A workflow system comprises a collection of workflow authorization schemata and a reference monitor. Each workflow schemata may be instantiated and executed as a sequence of tasks, which we will refer to as an *instance* of the workflow. We assume that the reference monitor will ensure that only authorized users perform tasks and that it will ensure all authorization constraints for a particular schema are satisfied in each workflow instance of that schema. The existence of both authorization information and constraints means that the design and analysis of reference monitors in workflow systems is rather more difficult than for similar mechanisms in computer file systems or relational database management systems, which usually make decisions based solely on authorization information. We believe that there are three distinct, but related, problems in workflow systems.

1. Is the constrained workflow specification satisfiable? In other words, is it possible for a workflow instance based on this specification to be completed so that all constraints are satisfied? Note that this question is posed without reference to authorization information. A trivial example of a workflow specification that is not satisfiable contains two tasks t and t' and two constraints: no user can perform t and t' (separation of duty) and the same user must perform t and t' (binding of duty).

2. Is the workflow authorization schema satisfiable? In other words, is it possible for a workflow instance based on this schema to be completed? A trivial example of a schema that is not satisfiable contains a task t which no user is authorized to execute.

3. Is a workflow instance in a workflow system satisfiable? In other words, given that a number of tasks in a workflow specification have been completed, can the remaining tasks be completed and the authorization constraints be satisfied?

Most research in this area has focused on the third of these questions. Solutions to this question have typically involved constructing a reference monitor that enforces authorization constraints. In this paper we adopt a formal, algebraic approach to the first of these questions and demonstrate how it can be used to answer the remaining questions and how it can help in the design of a reference monitor for workflow systems.

The difficulty of such issues is compounded because there is no consensus on the specification and scope of authorization constraints. In particular, different authors consider different types of constraints and different methods of implementing a reference monitor. Bertino *et al*, for example, consider cardinality constraints and role-based constraints and implement a reference monitor using logic programming techniques [3], whereas Casati *et al* consider binding of duty constraints and use active database technology to enforce constraints [7].

In order to provide a clear analysis of workflow systems and to design a reference monitor for such systems, it is vital to have a sound and unambiguous interpretation of authorization constraints. In this paper we claim that many of the constraints that have been considered in the literature are special cases of a general type of constraint, which we call an *entailment* constraint. We also argue that role-based constraints of the form "task t_2 must be performed by a role that is more senior that the role that performed task t_1" are not well defined and suggest how they can be recast either as a constraint on authorization information or as a user-based constraint. We also demonstrate that cardinality constraints can be specified as entailment constraints.

In this paper we clarify precisely what types of authorization constraints should be considered in workflow systems and how to interpret such constraints. We propose a powerful, simple, intuitive method for defining entailment constraints and illustrate how they can be used to specify separation of duty constraints, binding of duty constraints and cardinality constraints. In doing this we hope to provide a standard reference model for future research into constraints in workflow systems, thereby simplifying the task of designing and analyzing reference monitors for such systems.

Our scheme, described in Section 2, is based on binary relations defined on the set of users. Concepts from relation algebra can be used to combine relations and hence derive compound constraints on tasks. In this section we also introduce the important concept of an *execution schedule*, an abstraction of a workflow instance. In Section 3 we develop an algebra for entailment constraints that enables us to simplify and combine constraints. This gives rise to the notion of the *closure* of a set of authorization constraints. This in turn gives rise to a new workflow schema that is satisfiable if and only if the original schema is satisfiable. In Section 4 we describe how this powerful result enables us to develop new methods for analyzing the satisfiability of workflow schemata and workflow instances.

It is important to note that the lack of space prevents this paper from making a significant contribution to the design or implementation of a role-based reference monitor for workflow systems that incorporate authorization constraints. Important work has already been done in this area using stratified logic programs [3, 14] and active database technology [7]. Nevertheless, we believe that the implementation details in such research have sometimes obscured the problem at hand and have considered different notions of authorization constraints, making a comparative analysis of different approaches rather difficult.

Our thesis is that a formal definition of authorization constraints and the satisfaction of such constraints in a workflow instance, coupled with a rigorous analysis of authorization constraints, will lead to improved models for role-based reference monitors in workflow systems. Indeed, our work on the unsatisfiability of a set of constraints already suggests alternative ways of implementing a reference monitor that enforces authorization constraints. Developing a model for such a reference monitor will be our immediate priority in future work, which is discussed more fully in Section 5.

2 A Model for Constrained Workflow Systems

A *workflow specification* is a partially ordered set of tasks T. (We use T to denote the set of tasks in the specification. We will write t to denote a task in a workflow instance corresponding to the task $\mathsf{t} \in \mathsf{T}$.) Let U be a set of users. A *workflow authorization schema* is a pair (T, A), where A contains sufficient information to enable us to derive a relation $TU \subseteq \mathsf{T} \times U$ and $(\mathsf{t}, u) \in TU$ means that u is *authorized to perform* (or *execute*) t. A could comprise, for example, a partially ordered set of roles R, a user-role assignment relation $UA \subseteq U \times R$ and a task-role assignment relation $TA \subseteq \mathsf{T} \times R$. We now consider the specification of authorization constraints in a workflow schema.

2.1 Entailment Constraints

We believe that a specification scheme for authorization constraints should have the following properties.

- It must be unambiguous.
- It must provide sufficient expressive power to capture a wide range of security requirements including separation of duty constraints, binding of duty constraints and cardinality constraints.
- It must be simple enough to be used by application designers and departmental administrators.
- It must be independent of the underlying workflow and the reference monitor.
- It must be amenable to analysis.

In this section we describe our specification scheme, which is based on binary relations defined on the set of users. Such relations are expressive, intuitive and can be manipulated algebraically, enabling us to derive new constraints that simplify the analysis of workflows.

Let $Rel(U)$ denote the set of all binary relations on U. (In other words, $Rel(U)$ is the powerset of $U \times U$.) We use the following notation, which is standard in the study of relation algebras [11].

$$0 = \emptyset \tag{1}$$
$$0' = \{(u, v) : u, v \in U, u \neq v\} \tag{2}$$
$$1' = \{(u, u) : u \in U\} \tag{3}$$
$$1 = 1' \cup 0' = U \times U \tag{4}$$

An *entailment constraint* has the form $(D, (\mathsf{t}, \mathsf{t}'), \rho)$, where $D \subseteq U$, $\rho \in Rel(U)$ and $\mathsf{t} \not\geqslant \mathsf{t}'$. A *constrained workflow authorization schema* is a triple (T, A, C), where C is a set of entailment constraints.

Informally, if users u and u' perform $\mathtt{t}$ and $\mathtt{t}'$, respectively, and $u \in D$, then constraint $(D, (\mathtt{t}, \mathtt{t}'), \rho)$ is satisfied iff $(u, u') \in \rho$. In other words, the constraint is not applied if $u \notin D$. We refer to D as the *domain* of the constraint. We will define constraint satisfaction formally in the next section.

Hence a separation of duty constraint can be expressed as $(D, (\mathtt{t}, \mathtt{t}'), 0')$ and a binding of duty constraint can be expressed as $(D, (\mathtt{t}, \mathtt{t}'), 1')$. Henceforth, we will usually write $(D, (\mathtt{t}, \mathtt{t}'), \neq)$ instead of $(D, (\mathtt{t}, \mathtt{t}'), 0')$ and $(D, (\mathtt{t}, \mathtt{t}'), =)$ instead of $(D, (\mathtt{t}, \mathtt{t}'), 1')$, because it is more intuitive notation. However, in order to maintain clarity, we will use $0'$ and $1'$ when we form compound binary relations.

In fact, any binary relation between users can be used in an entailment constraint (including those that can be derived from contextual information). Hence it is possible to articulate constraints of the form "tasks $\mathtt{t}$ and $\mathtt{t}'$ must be performed by two different users in the same department". If we assume the existence of group-based or role-based authorization structures, then it is possible to induce an ordering (binary relation) on the set of users determined by the relative seniority of the groups or roles to which each user is assigned. The relation $l \in Rel(U)$ will be used to denote a partial ordering $\leqslant$ on users, which may be derived, depending on context, from role information, organizational information or the user groups to which users belong.

We anticipate that this relation will prove particularly important, because it is natural to implement access control in workflow systems using role-based techniques. Briefly, each task and user is assigned to one or more roles and a user u is authorized to perform a task $\mathtt{t}$ if u if u and $\mathtt{t}$ are both assigned to the same role. The administration of such systems can be greatly reduced if a role hierarchy is employed. Then u is authorized to perform a task $\mathtt{t}$ if u is implicitly assigned to a role to which $\mathtt{t}$ is also assigned. In other words, u is assigned to a role at least as senior as a role to which $\mathtt{t}$ is assigned. We examine practical applications of this in Section 2.3.

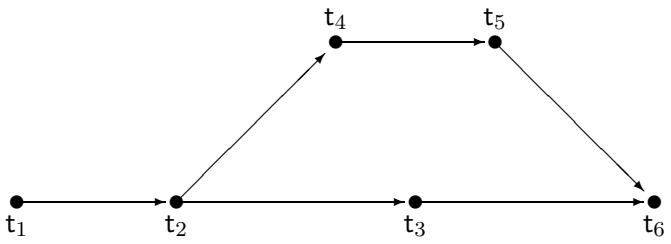
2.2 Linear Extensions of a Workflow Specification

Let $\langle X, \leqslant \rangle$ be a partially ordered set. A *linear extension* of X is a total ordering of the elements of X that respects the ordering of the elements in X. In other words, $\langle X, \preccurlyeq \rangle$ is a linear extension of $\langle X, \leqslant \rangle$ if for all $x_1, x_2 \in X$, either $x_1 \preccurlyeq x_2$ or $x_2 \preccurlyeq x_1$, and if $x_1 \leqslant x_2$ then $x_1 \preccurlyeq x_2$.[1] We denote the set of linear extensions of X by $\mathcal{L}(X)$.

Linear extensions are important in the context of workflows because they "linearize" a partially ordered set of tasks. In other words, a linear extension of T represents a possible sequence of execution of the tasks in a workflow. Figure 1 shows a simple example of a workflow specification having three linear extensions.

Definition 1 *Let (T, A, C) be a constrained workflow authorization schema. An* execution schedule *for (T, A, C) is a pair (L, α), where $L \in \mathcal{L}(\mathsf{T})$ and $\alpha : \mathsf{T} \to U$ assigns tasks to users, such that for all $\mathtt{t} \in \mathsf{T}$, $(\mathtt{t}, \alpha(\mathtt{t})) \in A$, and for all $(D, (\mathtt{t}, \mathtt{t}'), \rho) \in C$, if $\alpha(\mathtt{t}) \in D$ then $(\alpha(\mathtt{t}), \alpha(\mathtt{t}')) \in \rho$.*

[1] Linear extensions are obtained by topologically sorting X [10].



(a)

$\mathsf{t}_1 \prec \mathsf{t}_2 \prec \mathsf{t}_3 \prec \mathsf{t}_4 \prec \mathsf{t}_5 \prec \mathsf{t}_6$

$\mathsf{t}_1 \prec \mathsf{t}_2 \prec \mathsf{t}_4 \prec \mathsf{t}_3 \prec \mathsf{t}_5 \prec \mathsf{t}_6$

$\mathsf{t}_1 \prec \mathsf{t}_2 \prec \mathsf{t}_4 \prec \mathsf{t}_5 \prec \mathsf{t}_3 \prec \mathsf{t}_6$

(b)

Figure 1: A simple workflow specification and its linear extensions

In other words, an execution schedule respects the relative ordering of tasks in the workflow specification (since it is a linear extension of T), every task is performed by an appropriately authorized user and every entailment constraint is satisfied. A constrained workflow authorization schema is *satisfiable* if there exists an execution schedule for the schema (and *unsatisfiable* otherwise).

2.3 Other Constraints in Workflow Systems

2.3.1 Role-based entailment constraints

Constraints of the form "t_2 must be performed by a role that is more senior than the role that performed t_1" have received attention in the literature [2, 3]. It seems superficially attractive to extend the specification scheme for entailment constraints to include role-based ones of the form $(S, \{\mathsf{t}, \mathsf{t}'\}, pred)$, where $S \subseteq R$. However, we believe that such constraints are inappropriate in the wider context of role-based access control and should generally be expressed as constraints on the authorization schema. The constraint "t_2 must be performed by a role that is more senior than the role that performed t_1", for example, could be re-formulated as "for every role to which task t_1 is assigned, there must be a more senior role to which t_2 is assigned". An advantage of this approach is that such constraints can be enforced once by the authorization information, rather than in each instance of the workflow. In the remainder of this section, we justify more fully why we believe role-based entailment constraints should not be used.

In order for a user u to perform task t, u must be assigned to some role r and the task must be assigned to some role r' such that $r' \leqslant r$.However, the answer to the question "Which role performed task t?" is ambiguous. There are two obvious answers to this question: either r because this role implicitly assigns u to r', the role that is assigned to t; or r' because it is the role that is assigned to task t. Nevertheless, both of these interpretations have their problems. In the first case, what happens if u is assigned to two roles r_1 and r_2 that are both greater than r' and r_1 and r_2 are not comparable in the role hierarchy? Which of r_1 and r_2 is considered to be the role that performed t? In the second case, what happens if t is assigned to two roles r'_1 and r'_2 that are both less than

r? Which of r'_1 and r'_2 is considered to be the role that performed t? In order to address these problems, we consider two simplifying assumptions.

Assumption 1 Every task is assigned to precisely one role. *Therefore, we can assume that the role that performed the task is the role to which it is explicitly assigned. This means that any role-based constraints can be checked statically. For example, to enforce an entailment constraint of the form* $(R, \{\mathtt{t}, \mathtt{t}'\}, <)$*, it is sufficient to check that* $\mathtt{t}$ *and* $\mathtt{t}$ *are assigned to roles* r *and* r'*, respectively, with* $r < r'$*.*

Assumption 2 Every user is assigned to precisely one role. *We then assume that the role that performed a task is the role to which the user who performed the task is assigned. In this case, it is not possible check role-based constraints statically.*

Either way, it seems clear that role-based entailment constraints of the form "$\mathtt{t}_2$ must be performed by a role that is more senior than the role that performed $\mathtt{t}_1$" are at best ambiguous unless we make one of the two assumptions described above. Unfortunately, imposing such limitations on a role-based authorization scheme rather dilutes the power of the RBAC paradigm. The second assumption, in particular, reduces the access control mechanism to one that is equivalent to Unix groups. In short, either role-based entailment constraints cannot be interpreted or they require simplifications to the RBAC model that compromise the advantages provided by the RBAC paradigm.

Finally, we note the following quote: "... if several roles are authorized to execute a task and no order for those roles is specified ... the task can be performed by any of the roles" [3]. We would argue that this alone is a strong justification for believing that it is not possible in general to state unambiguously which role performed a given task.

An alternative approach is to state conditions on the assignment of roles to tasks. Henceforth, constraints of the form "$\mathtt{t}_2$ must be performed by a role that is more senior than the role that performed $\mathtt{t}_1$" will be implemented by making appropriate task-role assignments (as suggested in the opening paragraph of this section). Clearly, the satisfaction of this constraint can be determined by an analysis of the workflow authorization schema.

2.3.2 Additional user-based entailment constraints

Nevertheless, we are left with the problem of implementing security requirements of the following form: "$\mathtt{t}_2$ must be performed by a *user* who is more senior than the *user* who performed $\mathtt{t}_1$". Consider the following scenario: a workflow includes tasks that prepare a cheque and approve the release of the cheque to the payee. The preparation of the cheque is assigned to a clerical role, while the approval is assigned to a managerial role. The semantics of role-based access control mean that a user assigned to a role r can perform any task assigned to a role r' such that $r' \leqslant r$. Hence a manager (acting in a managerial role) can prepare a cheque payment and part of our security policy may require that such a cheque has to be approved by a senior manager. Certainly, user-based entailment constraints can be used to prevent the same user preparing and approving a cheque payment, but how can we implement the security requirement described above?

The answer is to define binary relations based on the role hierarchy and the assignment of users to roles. Let $R(u)$ denote the set of roles assigned to a user u. That is, $R(u) = \{r \in R : r \leqslant r', (u, r') \in UA\}$. We define the following relations.

$$e = \{(u, v) : u, v \in U, R(u) = R(v)\} \tag{5}$$
$$l = \{(u, v) : u, v \in U, R(u) \subseteq R(v)\} \tag{6}$$
$$g = \{(u, v) : u, v \in U, R(u) \supseteq R(v)\} = \widetilde{l} \tag{7}$$

Essentially, the set of roles assigned to each user and the role hierarchy induce an ordering on the set of users. A user u is less senior than another user u' if u' is assigned to all the roles to which u is assigned. Note that l is not a partial ordering on the set of users, since it is not anti-symmetric. In other words, it may be the case that $(u, v), (v, u) \in l$ but $u \neq v$. In this case, the users have *equivalent* authority (hence the use of e for the set of pairs (u, v) such that $R(u) = R(v)$).

To help the reader we will generally use the more familiar infix notation for relations in the body of constraints and write

$$u = u' \text{ if } (u, u) \in 1',$$
$$u \cong v \text{ if } (u, v) \in e,$$
$$u \leqslant v \text{ if } (u, v) \in l,$$
$$u \geqslant v \text{ if } (u, v) \in g.$$

We also write

$$u \neq v \text{ if } (u, v) \in 0',$$
$$u < v \text{ if } (u, v) \in l \setminus e,$$
$$u > v \text{ if } (u, v) \in g \setminus e.$$

However, we will tend to use the relation symbols when writing algebraic expressions such as $0' \cap 1' = \emptyset$ in order to avoid inelegant and rather cryptic statements of the form $\neq \cap == \emptyset$.

Figure 2 shows the ordering induced on a set of users by the role hierarchy and the UA relation. Note that $R(u) \subseteq R(v)$ if for every role r such that $(u, r) \in UA$, there exists r' such that $r \leqslant r'$ and $(v, r') \in UA$. Hence, `Alice` $<$ `Eve`, for example, because $(\texttt{Alice}, \texttt{FinAdmin}), (\texttt{Eve}, \texttt{FinAdmin}) \in UA$ and $(\texttt{Alice}, \texttt{POClerk}), (\texttt{Eve}, \texttt{POAdmin}) \in UA$ and `POClerk` $<$ `POAdmin`.

The analysis of the previous section suggested that role-based entailment constraints should be recast as constraints on the assignment of tasks to roles. In this section we have shown that security requirements regarding the relative seniority of user can be enforced by defining entailment constraints with suitable binary relations. In the next section we consider how cardinality constraints can be expressed as entailment constraints. Hence we will have shown that many of the authorization constraints in the literature can be expressed as entailment constraints. However, we note that we cannot specify constraints

of the following form: "If more that four activations of task $\mathtt{t}_1$ executed by a single user within a single instance of a workflow abort then that user cannot execute task $\mathtt{t}_1$" [3]. In fact, we do not consider such constraints to be authorization constraints. We believe that such constraints have more in common with "lockout" policies for user accounts, which are used to disable an account if an incorrect password is entered too many times consecutively. These policies often have a number of parameters, including the number of fail attempts that are tolerated, and when and who may unlock an account. We believe that such policies should be addressed independently of authorization constraints.

We also note that using our framework it is possible to express constraints that have not been considered previously. For example, we can insist that two different tasks are performed by users who have equivalent powers using a constraint of the form $(D, (\mathtt{t}, \mathtt{t}'), \cong)$. We also note that it is easy to define constraints that can prevent collusion between family or friends working within the same enterprise by defining an appropriate binary relation encoding undesirable associations between pairs of users.

2.3.3 Cardinality constraints

Like role-based entailment constraints, cardinality constraints have received a considerable amount of attention in the literature [2, 3, 14]. Informally, a cardinality constraint requires that a task be executed a number of different times by a number of different users. Note that in order to enforce such constraints, we must know which user and role performed each instance of the task. In other words, there has to be some sequence of task instances and we can treat the k instances of $\mathtt{t}$ as distinct sequential tasks upon which entailment constraints are defined. In the remainder of this section, we demonstrate that task-based cardinality constraints can be specified as entailment constraints.

A task-based cardinality constraint can be modelled as a tuple $(\mathtt{t}, k, n_u, n_r)$. The interpretation of this constraint is that task $\mathtt{t}$ has to be performed k times by at least n_u different users and at least n_r different roles. If n_u (respectively n_r) is not specified, then there is no restriction on the users (roles) that can perform task $\mathtt{t}$.

Given the argument in Section 2.3.1, we do not believe that it is possible to identify which role performs an instance of a task. We note that the most comprehensive treatment of task-based cardinality constraints assumes that each instance of the same task is performed by the same role [3].

Hence we can assume that a task-based cardinality constraint has the form $(\mathtt{t}, k, n)$. There are three cases to consider: n is not specified ($n = \mathtt{null}$), $n = k$ and $n < k$. In the first two cases we simply modify the workflow specification to include k tasks $\mathtt{t}_1 < \cdots < \mathtt{t}_k$ instead of the single instance of $\mathtt{t}$. In order to enforce the cardinality constraint $(\mathtt{t}, k, k)$ we define entailment constraints $(U, \{\mathtt{t}_i, \mathtt{t}_j\}, \neq)$ $(1 \leqslant i < j \leqslant k)$ for the modified workflow. The cardinality constraint $(\mathtt{t}, k, \mathtt{null})$ does not require any entailment constraints in the modified workflow. In order to implement a cardinality constraint of the form $(\mathtt{t}, k, n)$, where $n < k$, we define k tasks $\mathtt{t}_1, \ldots, \mathtt{t}_k$ such that $\mathtt{t}_1 < \cdots < \mathtt{t}_n$ and $\mathtt{t}_1 < \mathtt{t}_{n+1} < \mathtt{t}_n$, $\mathtt{t}_1 < \mathtt{t}_{n+2} < \mathtt{t}_n$, $\ldots$, $\mathtt{t}_1 < \mathtt{t}_k < \mathtt{t}_n$. In addition we define entailment constraints $(U, \{\mathtt{t}_i, \mathtt{t}_j\}, \neq)$, $1 \leqslant i < j \leqslant n$, to ensure that at least n different users perform the tasks in $\{\mathtt{t}_1, \ldots, \mathtt{t}_k\}$.

We do not consider workflow-based cardinality constraints of the form "at least three roles must be involved in completing a workflow" [3]. As before, we believe such a constraint should be enforced by appropriate task-role assignments. However, a constraint of the form "at least three users must be involved in completing a workflow" cannot be enforced using entailment constraints (because it is not obvious which tasks should be antecedent tasks). We are not aware, however, of any attempts to specify or enforce such constraints.

2.4 An example of a constrained workflow schema

Let us consider the simple workflow forming part of a purchase ordering and financial system depicted in Figure 3(a). There are six tasks involved in ordering and paying for goods:

- the creation of a purchase order requesting goods from a supplier (`createPO`);
- the approval of the purchase order prior to despatch to the supplier (`apprPO`);
- the acknowledgement of delivery of the goods by signing a goods received note (`signGRN`);
- the acknowledgement of delivery by countersigning the goods received note (`ctrsignGRN`);
- the creation of a payment file on receipt of the supplier's invoice for the goods (`createPay`);
- the approval of the payment to the supplier (subject to receipt of goods) (`apprPay`).

The entailment constraints for these tasks together with a brief explanation are shown in Figure 3(b). We believe that these constraints form a realistic set of security requirements for such a workflow. Note that because `createPay` and `signGRN` are not comparable in the workflow specification, we need two constraints, c_5 and c_6, to prevent a user raising a payment for goods that he has signed for.

3 The Algebra of Entailment Constraints

Let $\rho, \sigma \in Rel(U)$ and let $V \subseteq U$. Then we define

$$\widetilde{\rho} = \{(v,u) : (u,v) \in \rho\}, \tag{8}$$

$$\rho\sigma = \{(u,w) : \exists v \in U, (u,v) \in \rho, (v,w) \in \sigma\}, \tag{9}$$

$$U_1(\rho) = \{u \in U : (u,v) \in \rho\}, \tag{10}$$

$$U_2(\rho) = \{v \in U : (u,v) \in \rho\}, \tag{11}$$

$$\overline{V} = \{u \in U : u \notin V\}. \tag{12}$$

We say $\widetilde{\rho}$ is the *converse* of ρ and $\rho\sigma$ is the *relative product* or *composition* of ρ and σ. We have the following simple results for all $\rho \subseteq 1$:

$$\begin{aligned}
&1\rho = U \times U_2(\rho) \quad \text{and} \quad \rho 1 = U_1(\rho) \times U,\\
&\rho 1' = 1'\rho = \rho,\\
&\rho 0 = 0\rho = 0,\\
&1\rho = \rho 1,
\end{aligned}$$

and

$$11' = 1'1 = 10' = 0'1 = 0'0' = 1.$$

Proposition 2 (Merging domains) *Let* $(\mathsf{T}, A, \{(D_1, (\mathsf{t}, \mathsf{t}'), \rho), (D_2, (\mathsf{t}, \mathsf{t}'), \rho)\}$ *be a constrained workflow authorization schema. Then* (L, α) *is a workflow execution schedule for* $(\mathsf{T}, A, \{(D_1, (\mathsf{t}, \mathsf{t}'), \rho), (D_2, (\mathsf{t}, \mathsf{t}'), \rho)\}$ *iff* (L, α) *is a workflow execution schedule for* $(\mathsf{T}, A, \{D_1 \cup D_2, (\mathsf{t}, \mathsf{t}'), \rho\})$.

Proof The proof is immediate. ■

In other words, we can assume that there is at most one constraint for a given (ordered) pair of tasks $(\mathsf{t}, \mathsf{t}')$ and a given binary relation ρ.

Proposition 3 (Expanding the domain) *Let* $(\mathsf{T}, A, \{(D, (\mathsf{t}, \mathsf{t}'), \rho)\})$ *be a constrained workflow authorization schema and define* $\sigma = (U \setminus D) \times U$. *Then* (L, α) *is a workflow execution schedule for* $(\mathsf{T}, A, \{(D, (\mathsf{t}, \mathsf{t}'), \rho)\})$ *iff* (L, α) *is a workflow execution schedule for* $(\mathsf{T}, A, \{(U, (\mathsf{t}, \mathsf{t}'), \rho \cup \sigma)\})$.

Proof If (L, α) is an execution schedule for $(\mathsf{T}, A, \{D, (\mathsf{t}, \mathsf{t}'), \rho\})$ then either $\alpha(\mathsf{t}) \in D$ and $(\alpha(\mathsf{t}), \alpha(\mathsf{t}')) \in \rho$, or $\alpha(\mathsf{t}) \notin D$, in which case $(\alpha(\mathsf{t}), \alpha(\mathsf{t}')) \in \sigma$, by definition of σ. In either case, constraint $(U, (\mathsf{t}, \mathsf{t}'), \rho \cup \sigma)$ is satisfied.

If (L, α) is an execution schedule for $(\mathsf{T}, A, \{(U, (\mathsf{t}, \mathsf{t}'), \rho \cup \sigma)\})$ then either $\alpha(\mathsf{t}) \in D$, in which case $(\alpha(\mathsf{t}), \alpha(\mathsf{t}')) \in \rho$, by definition of σ, and $(D, (\mathsf{t}, \mathsf{t}'), \rho)$ is satisfied; otherwise $\alpha(\mathsf{t}) \notin D$ and $(D, (\mathsf{t}, \mathsf{t}'), \rho)$ is vacuously satisfied. ■

In other words, we can assume that the domain of every constraint is U. Henceforth, we will omit the domain from our constraints.

Proposition 4 (Merging constraints) *Let* $W = (\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho_1), ((\mathsf{t}, \mathsf{t}'), \rho_2)\})$ *be a constrained workflow authorization schema. Then* (L, α) *is an execution schedule for* W *iff* (L, α) *is an execution schedule for* $(\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho \cap \rho_2)\})$.

Proof The proof is immediate. ■

In other words, we can assume that for each pair of tasks $(\mathsf{t}, \mathsf{t}')$ there is a single constraint. (If two constraints contain the same pair of tasks, then we can replace them with a single constraint using the proposition above.)

Proposition 5 (Composing constraints) *Let* $W = (\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho_1), ((\mathsf{t}', \mathsf{t}''), \rho_2)\})$ *be a constrained workflow authorization schema. Then* (L, α) *is an execution schedule for* W *iff* (L, α) *is an execution schedule for* $(\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho_1), ((\mathsf{t}', \mathsf{t}''), \rho_2), ((\mathsf{t}, \mathsf{t}''), \rho_1\rho_2)\})$.

Proof Let (L, α) be an execution schedule for W. Then $(\alpha(\mathsf{t}), \alpha(\mathsf{t}')) \in \rho_1$ and $(\alpha(\mathsf{t}'), \alpha(\mathsf{t}'')) \in \rho_2$. Hence $(\alpha(\mathsf{t}), \alpha(\mathsf{t}'')) \in \rho_1\rho_2$ and $((\mathsf{t}, \mathsf{t}''), \rho_1\rho_2)$ is satisfied. The proof is trivial in the other direction. ■

It is important to note that if (L, α) is an execution schedule for $(\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}''), \rho_1\rho_2)\})$, then it is not necessarily an execution schedule for $(\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho_1), ((\mathsf{t}', \mathsf{t}''), \rho_2)\})$. (Although we have $(\alpha(\mathsf{t}), \alpha(t'')) \in \rho_1\rho_2$, we can not necessarily infer that there exists an authorized user for t'.) In other words, we cannot delete the constraints from which a compound constraint is derived.

4 Analyzing Workflows

In this section we discuss the analysis of constrained workflow schemata. Space does not permit a detailed and formal discussion, which we will be left for future work.

Given a constrained workflow specification, we can enumerate all possible linear extensions and apply the entailment constraints. In general, the generation of the set of linear extensions in T can be performed in time $\mathcal{O}(|\mathcal{L}(\mathsf{T})|)$ [12] and computing $|\mathcal{L}(\mathsf{T})|$ is #P-complete [5]. However, if the width of the poset is small (as will be the case for a typical workflow specification), then the set of linear extensions can be computed quickly using dynamic programming techniques. (Each linear extension is a directed path in the graph of the lattice of order ideals of T, in which the number of nodes is bounded by $|\mathsf{T}|^w$, where w is the width of T. The directed paths can be computed using a breadth-first search using a number of operations linear in the number of nodes of the graph [13]. In our example $w = 2$.)

For each linear extension we compute every implied constraint and then form a single constraint for each pair of tasks in the linear extension by taking the intersection of all relations that apply for that pair of tasks. Let C^* denote the set of constraints obtained from C in this way. We refer to C^* as the *closure* of C.

Theorem 6 (L, α) *is an execution schedule for* (T, A, C) *iff* (L, α) *is an execution schedule for* (T, A, C^*).

Proof Every constraint $c \in C^*$ is the intersection of one or more constraints of the form $c_1c_2 \ldots c_m$, where $m < |\mathsf{T}|$. The results in Propositions 4 and 5 can be generalized using induction to the intersection and composition of $k \geqslant 2$ relations. The result now follows. ■

The constraints in the closure of the set of constraints for the purchase order workflow are shown below. For clarity, we use ℓ to denote the binary relation $<$. The constraints marked with an asterisk appear in the original set of constraints.

$$
\begin{array}{l}
((\texttt{createPO}, \texttt{apprPO}), \ell)^* \\
((\texttt{createPO}, \texttt{createPay}), \ell\texttt{1} \cap \texttt{0}' \cap \ell\texttt{10}' \cap \ell\texttt{10}'\texttt{0}') \\
((\texttt{createPO}, \texttt{signGRN}), \texttt{1}' \cap \ell\texttt{1} \cap \ell\texttt{10}') \\
((\texttt{createPO}, \texttt{ctrsignGRN}), \texttt{0}' \cap \ell\texttt{10}' \cap \ell\texttt{10}'\texttt{0}') \\
((\texttt{createPO}, \texttt{apprPay}), \ell \cap \ell\texttt{0}' \cap \ell\texttt{1}\ell \cap \texttt{0}'\ell \cap \ell\texttt{10}'\ell \cap \ell\texttt{10}'\texttt{0}'\ell) \\
((\texttt{apprPO}, \texttt{apprPay}), \texttt{0}' \cap \ell\texttt{1}) \\
((\texttt{createPay}, \texttt{signGRN}), \texttt{0}')^* \\
((\texttt{createPay}, \texttt{apprPay}), \ell)^* \\
((\texttt{signGRN}, \texttt{createPay}), \texttt{0}')^* \\
((\texttt{signGRN}, \texttt{ctrsignGRN}), \texttt{0}')^* \\
((\texttt{signGRN}, \texttt{apprPay}), \texttt{0}'\ell)^* \\
((\texttt{ctrsignGRN}, \texttt{apprPay}), \ell\texttt{1})
\end{array}
$$

It can easily be shown that $a\texttt{10}' = a\texttt{1}$ for any $a \subseteq U \times U$ and since $a \subseteq b$ implies $a \cap b = a$, we obtain

$$
\begin{array}{l}
((\texttt{createPO}, \texttt{apprPO}), \ell)^* \\
((\texttt{createPO}, \texttt{createPay}), \ell\texttt{1} \cap \texttt{0}') \\
((\texttt{createPO}, \texttt{signGRN}), \texttt{1}' \cap \ell\texttt{1}) \\
((\texttt{createPO}, \texttt{ctrsignGRN}), \texttt{0}' \cap \ell\texttt{1}) \\
((\texttt{createPO}, \texttt{apprPay}), \ell \cap \ell\texttt{0}' \cap \ell\texttt{1}\ell \cap \texttt{0}'\ell) \\
((\texttt{apprPO}, \texttt{apprPay}), \texttt{0}' \cap \ell\texttt{1}) \\
((\texttt{createPay}, \texttt{signGRN}), \texttt{0}')^* \\
((\texttt{createPay}, \texttt{apprPay}), \ell)^* \\
((\texttt{signGRN}, \texttt{createPay}), \texttt{0}')^* \\
((\texttt{signGRN}, \texttt{ctrsignGRN}), \texttt{0}')^* \\
((\texttt{signGRN}, \texttt{apprPay}), \texttt{0}'\ell)^* \\
((\texttt{ctrsignGRN}, \texttt{apprPay}), \ell\texttt{1})
\end{array}
$$

Corollary 7 *Let* (T, A, C) *be a constrained workflow authorization schema. If* $((\mathsf{t}, \mathsf{t}'), \emptyset) \in C^*$ *then* (T, A, C) *is unsatisfiable.*

Proof The existence of such a constraint implies that no user can perform t (and hence the workflow (T, A, C^*) is unsatisfiable). The result follows by Theorem 6. ■

Hence we can perform a preliminary analysis of a constrained workflow specification, without any reference to authorization data. If A is known, then we can also compute all users that should be allowed to perform tasks if the constraints in C^* are to be satisfied. This analysis will perform two purposes: firstly, it will enable us to determine whether the workflow schema is satisfiable; secondly, it will enable us to modify the authorization data in order to simplify the design of the reference monitor.

4.1 Determining whether a workflow schema is satisfiable

Let $W = (\mathsf{T}, A, C)$ be a constrained workflow authorization schema. For every task $\mathsf{t} \in \mathsf{T}$ we compute (using the information in A) the set of users authorized to perform t, denoted $U(\mathsf{t})$. If A is simply a subset of $\mathsf{T} \times U$ that explicitly assigns tasks to users, then $U(\mathsf{t}) = \{u \in U : (\mathsf{t}, u) \in A\}$. If A contains information about the assignment of users and tasks to roles, then we must realize the assignment of tasks to users as the composition of the relations TA and $\widetilde{UA} = \{(r, u) : (u, r) \in UA\}$. If, in addition, A contains a role hierarchy, then we realize the assignment of tasks to users as the composition of TA, RH and $\widetilde{UA}$, where $(r, r') \in RH$ iff $r \leqslant r'$ in the role hierarchy.

Then for every constraint $((\mathsf{t}, \mathsf{t}'), \rho) \in C^*$ we compute whether there are users authorized to perform t and t' and who satisfy the constraint. In particular, if $(U(\mathsf{t}) \times U(\mathsf{t}')) \cap \rho = \emptyset$, then there is no pair of users that are authorized to execute the tasks and simultaneously satisfy the constraint and hence there cannot exist an execution schedule for (T, A, C^*). Therefore, assuming that the workflow specification itself is satisfiable, we must amend A so that suitably authorized users exist.

4.2 Revising the assignment of tasks to users

We can undertake a more detailed analysis of a workflow authorization schema to determine if certain users should be prevented from executing certain tasks because of the subsequent impossibility of satisfying an authorization constraint. In particular, for each constraint $((\mathsf{t}, \mathsf{t}'), \rho) \in C^*$ and for each user in $U(\mathsf{t})$ we compute $(\{u\} \times U(\mathsf{t}')) \cap \rho$.[2] If this evaluates to $\emptyset$ then no user can perform t' if u performs t. Hence, we should delete (t, u) from A. Unfortunately, this is non-trivial if we are using role-based techniques and is even more difficult if a role hierarchy is being employed. One possible solution is to explicitly associate tasks that a user is prohibited from executing with the user. The reference monitor then allows a request (t, i, u) iff (t, u) is authorized by A and u is not prohibited from performing task t.

[2]In fact this analysis can be performed by computing $U(\mathsf{t}, \mathsf{t}') = (U(\mathsf{t}) \times U(\mathsf{t}')) \cap \rho$ and determining which users appear in $U(t)$ but not in $U_1(\mathsf{t}, \mathsf{t}')$ (the projection onto the first element of $U(\mathsf{t}, \mathsf{t}')$). This can be realized as a relatively simply query in a relational database.

4.3 Static analysis of idealized users

A role can be regarded as a set of tasks. Hence, for each subset in $S \subseteq \mathsf{T}$ we can create the "role" r_S. A role can also be regarded as a collection of users. Hence we can regard each role r_S as a representative of the set of users assigned to the tasks in S. In other words, we can conduct an analysis of the workflow authorization schema $(\mathsf{T}, 2^{\mathsf{T}}, C^*)$, where 2^{T} is the powerset of tasks, and we assume that there is precisely one idealized user for each set of tasks.

Clearly, there exists a natural ordering on this set of roles: namely, $r_S \leqslant r_{S'}$ iff $S \subseteq S'$. Indeed, it is reasonable to use this approach for engineering roles and the hierarchy to which they belong. It may well be the case that not every subset of tasks constitutes a natural role in the organization and hence we need only consider subsets of T that actually make sense in the context of the organization and the workflow. In our running example, we might identify the following sets of tasks as potential roles in the workflow system.

$r_{\{\texttt{createPO}\}}$	`POClerk` role
$r_{\{\texttt{createPay}\}}$	`FinClerk` role
$r_{\{\texttt{createPO},\texttt{apprPO},\texttt{signGRN},\texttt{ctrsignGRN}\}}$	`POAdmin` role
$r_{\{\texttt{createPay},\texttt{apprPay},\texttt{signGRN},\texttt{ctrsignGRN}\}}$	`FinAdmin` role
$r_{\{\texttt{createPO},\texttt{apprPO},\texttt{createPay},\texttt{signGRN},\texttt{ctrsignGRN}\}}$	`FinAdmin` and `POClerk` roles
$r_{\{\texttt{createPay},\texttt{apprPay},\texttt{createPO},\texttt{signGRN},\texttt{ctrsignGRN}\}}$	`POAdmin` and `FinClerk` roles
$r_{\{\texttt{createPay},\texttt{apprPay},\texttt{createPO},\texttt{apprPO},\texttt{signGRN},\texttt{ctrsignGRN}\}}$	`FinAdmin` and `POAdmin` roles

Note that an analysis of the workflow authorization schema (T, A, C), where A associates each of these roles with the respective tasks, will show that the role $r_{\{\texttt{createPO}\}}$ should not be assigned to any user because if `createPO` is performed by role $r_{\{\texttt{createPO}\}}$, then it is not possible for task `signGRN` to be performed (because of the constraint $((\texttt{createPO}, \texttt{signGRN}), =)$). In other words, no role in the live workflow system should have the single task `createPO` assigned to it. Hence, the `POClerk` role should have (at least) the tasks `createPO` and `signGRN` assigned to it. Obviously, this is a simple example, but it is reasonable to suppose that in more complex workflows, such situations will not be easy to identify and that this method of finding potential authorization problems will prove very useful.

4.4 Implementing a Reference Monitor

To simplify the discussion, we assume that there is a single workflow authorization schema $W = (\mathsf{T}, A, C)$ in the system. We denote the ith instance of the schema by $W[i]$. (It will be clear that the techniques we describe can be extended to two or more authorization schemata.)

We assume that three sets of authorization information are maintained by the workflow system. Firstly, we maintain data structures that enable us to derive a binary relation $TU \subseteq \mathsf{T} \times U$ for each workflow specification. This is our authorization data A. In the

case of a role-based reference monitor, for example, A will consist of a role hierarchy, a user-role assignment relation and a task-role assignment relation. For convenience we will write $(\mathtt{t}, u) \in A$ if u is authorized to perform $\mathtt{t}$ (although A may not contain such a pair explicitly).

In our running example, $((\texttt{createPO}, \texttt{apprPay}), \ell \cap \ell 0' \cap \ell 1 \ell \cap 0' \ell)$ implies that neither `Eve` nor `Geoff` can execute `createPO` (because of the relation $\ell 1 \ell$) despite the fact that they are both authorized to perform the task. Hence, we also store additional (static) information prohibiting certain users from performing certain tasks. We will denote this set of data structures by B_S ('B' standing for "banned" and 'S' standing for "static").

In addition, we maintain data structures that enable us to derive a binary relation $-TU \subseteq \mathsf{T} \times \mathbb{N} \times U$, where $(\mathtt{t}, i, u) \in -TU$ implies that u is prohibited from performing task $\mathtt{t}$ in instance i of the workflow. We will denote this set of data structures by B_D ('D' standing for "dynamic'.) We will write $(\mathtt{t}, i, u) \in B$ if u is prohibited from performing $\mathtt{t} \in \mathsf{T}$ in $W[i]$ (although B may not contain such a pair explicitly). Some of this data may apply to every instance of a workflow.

The reference monitor will allow the request (t, i, u) in workflow $W[i]$ only if all of the following conditions are satisfied:

$$(\mathtt{t}, u) \notin B_S; \tag{13}$$

$$(\mathtt{t}, i, u) \notin B_D; \tag{14}$$

$$(\mathtt{t}, u) \in A. \tag{15}$$

After every task is performed, the reference monitor must also update B_D to include the users that cannot perform subsequent tasks because of the presence of an entailment constraint. Suppose $((\mathtt{t}, \mathtt{t}'), \neq) \in C$, for example, and that u performs t in instance i. Then $(\mathtt{t}', i, u)$ is added to the information in B_D. This kind of postprocessing is found in the implementations of both Bertino *et al* and Casati *et al*.

5 Related and Future Work

The work of Bertino *et al* guarantees that every workflow instance is satisfiable but this involves certain computational overheads, whereas Casati *et al* have a relatively efficient reference monitor but cannot guarantee that every workflow instance completes. As it stands, the reference monitor we have described lies somewhere between these two implementations. The computation of the closure of the set of constraints needs to be performed once, generates all possible constraints that exist between different tasks and is independent of authorization information. This makes it more efficient compared to other approaches. However, our reference monitor does not guarantee that all instances are satisfiable because the propagation of prohibitions may result in a subsequent task having no users that are both authorized and not banned from executing the task.

There are alternative methods of implementing a reference monitor given the information we have derived in C^*. For example, we could store each execution schedule for

(T, A, C^*). Each linear extension could refer to a table containing each possible set of user-task assignments for that linear extension. Given a workflow instance $W[i]$, where t_j has been performed by u_j $(1 \leqslant j \leqslant k)$, and a request (t_{k+1}, i, u), the reference monitor simply confirms that there exists at least one user-task assignment for each linear extension that begins $[t_1, \ldots, t_k, t_{k+1}]$. This guarantees that all constraints are satisfied, all users are appropriately authorized and that all constraints will be satisfied. However, if the set of users is large, this approach will be expensively computationally (as is the approach of Bertino *et al*). One advantage of this approach is that no processing is required after a request has been granted. A full analysis of this approach and a comparison with the work of Bertino *et al* [3] will the subject of future work.

A further advantage of this approach is that the linear extensions which provide information about the sequence in which tasks are performed and the users that can perform that sequence of tasks is decoupled. This means that if the user base changes, only the user sequences have to be re-computed. This is not the case for the user-role-assignment graph of Bertino *et al*. Moreover, because we do not consider entailment constraints that include roles, our analysis is independent of the assignment of roles to tasks (provided a user is acting in a suitably authorized role). Hence we are able to focus on user-based constraints and to consider constraining relationships between users that have been ignored hitherto.

There are also numerous opportunities for further theoretical work. Corollary 7 provides a necessary condition for the satisfiability of a workflow schema; it would be interesting to see what progress could made on providing a sufficient condition. We anticipate that existing results in model checking theory and relation algebras could help in this respect. We are also interested in finding an efficient way of computing the constraints in C^*. We are currently working on a technique that encodes constraints as the entries of an adjacency matrix for a graph derived from the partial order relation on T. Each constraint can be realized as an entry in some power of this adjacency matrix. We expect to announce progress on this issue very soon.

Acknowledgements We would like to thank Szabolcs Mikulas for several stimulating discussions on relation algebras and Frank Ruskey for his helpful remarks on generating linear extensions.

References

[1] G.-J. Ahn, R. Sandhu, M.H. Kang, and J.S. Park. Injecting RBAC to secure a web-based workflow system. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, pages 1–10, 2000.

[2] V. Atluri and W. Huang. An authorization model for workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security*, pages 44–64, 1996.

[3] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, 1999.

[4] R.A. Botha and J.H.P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.

[5] G. Brightwell and P. Winkler. Counting linear extensions. *Order*, 8:225–242, 1991.

[6] Burroughs. *Work Flow Management User's Guide*, 1973. Burroughs Manual 5000714.

[7] F. Casati, S. Castano, and M. Fugini. Managing workflow authorization constraints through active database technology. *Information Systems Frontiers*, 3(3):319–338, 2001. Technical Report HPL-2000-156, Hewlett Packard Laboratories.

[8] T. Jaeger and J. Tidswell. Practical safety in flexible access control models. *ACM Transactions on Information and System Security*, 4(2):158–190, 2001.

[9] K. Knorr and H. Stormer. Modeling and analyzing separation of duties in workflow environments. In *Trusted Information: The New Decade Challenge, IFIP TC11 Sixteenth Annual Working Conference on Information Security*, pages 199–212, 2001.

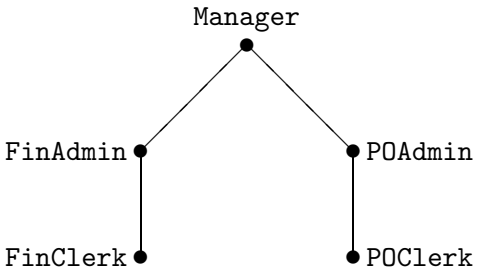
[10] D.E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1973.

[11] R.D. Maddux. Introductory course on relation algebras, finite-dimensional cylindric algebras, and their interconnections. In H. Andréka, J.D. Monk, and I. Németi, editors, *Algebraic Logic*, volume 54 of *Colloquia Mathematica Societatis János Bolyai*. János Bolyai Mathematical Society and Elsevier Science Publishers B.V., Amsterdam, 1991.

[12] G. Pruesse and F. Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, 23(2):373–386, 1994.

[13] F. Ruskey. Personal communication. 2004.

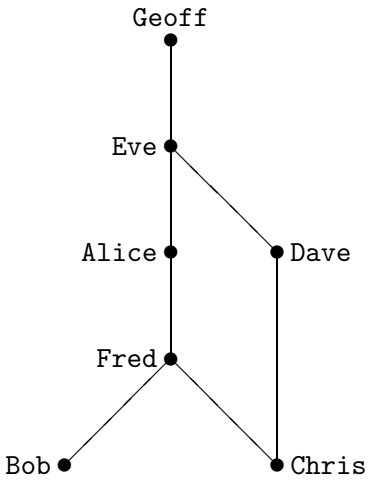
[14] J. Wainer, P. Barthelmess, and A. Kumar. W-RBAC – A workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems*, 12(4):455–486, 2003.



(a) The role hierarchy

UA	
User	**Role**
Alice	FinAdmin
Alice	POClerk
Bob	FinClerk
Chris	POClerk
Dave	POAdmin
Eve	FinAdmin
Eve	POAdmin
Fred	POClerk
Fred	FinClerk
Geoff	Manager

(b) The user-role assignment relation

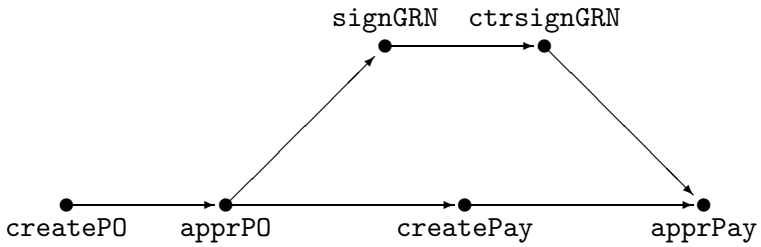


(c) The induced user hierarchy

$<$	
User1	**User2**
Alice	Eve
Alice	Geoff
Bob	Alice
Bob	Eve
Bob	Fred
Bob	Geoff
Chris	Alice
Chris	Dave
Chris	Eve
Chris	Fred
Chris	Geoff
Dave	Eve
Dave	Geoff
Eve	Geoff
Fred	Alice
Fred	Eve
Fred	Geoff

(d) The relation $l \setminus e$

Figure 2: Inducing an ordering on the set of users



(a) Workflow specification

c_1	$(U, (\texttt{createPO}, \texttt{apprPO}), <)$	The user that approves a purchase order must be more senior than the user that creates it
c_2	$(U, (\texttt{createPO}, \texttt{signGRN}), =)$	The user that creates a purchase order must sign for the goods
c_3	$(U, (\texttt{signGRN}, \texttt{ctrsignGRN}), \neq)$	The user that countersigns the GRN must be different from the user that signed the GRN
c_4	$(U, (\texttt{createPO}, \texttt{apprPay}), <)$	The user that approves the payment for goods must be more senior than the user that ordered the goods
c_5	$(U, (\texttt{signGRN}, \texttt{createPay}), \neq)$	The user that signs for the goods cannot create the payment for those goods
c_6	$(U, (\texttt{createPay}, \texttt{signGRN}), \neq)$	The user that creates the payment for the goods cannot sign for the goods
c_7	$(U, (\texttt{createPay}, \texttt{apprPay}), <)$	The user that approves the payment must be more senior than the user that creates the payment
c_8	$(U, (\texttt{apprPO}, \texttt{apprPay}), \neq)$	The user that approves the purchase order cannot approve the payment

(b) Entailment constraints

Figure 3: A simple example of a constrained workflow specification for a purchase order system